flash attention from math to code

wenqingqian May 2024

code see Github

Contents

1 online softmax

2 pass online softmax

for i in $[1, N]$:

$$\begin{aligned} m_i &= \max(x_i, m_{i-1}) \\ d'_i &= d'_{i-1} * e^{m_{i-1}-m_i} + e^{x_i-m_i} \end{aligned} \tag{1}$$

for i in $[1, N]$:

$$y_i = \frac{e^{x_i-m_N}}{d'_N} \tag{2}$$

In equation 1, each time we process a new number x_i, and calculate the new max number of sequence and summation. To make the algorithm parallel, we need to get the new result by two sequence directly, assuming there are two sequences:

$$
\begin{aligned}
A.\text{max} &= \text{max}(B.\text{max}, S.\text{max}) \\
A.\text{sum} &= S.\text{sum} \cdot e^{S.\text{max}-A.\text{max}} + B.\text{sum} \cdot e^{B.\text{max}-A.\text{max}}
\end{aligned} \tag{3}
$$

More detail this ariticle based

2 from online softmax to flash attention

attention with online softmax

for i in $[1, N]$:

$$
\begin{aligned}
x_i &= Q[k,:]K^T[:,i] \\
m_i &= \text{max}(x_i, m_{i-1}) \\
d'_i &= d'_{i-1} * e^{m_{i-1}-m_i} + e^{x_i-m_i}
\end{aligned} \tag{4}
$$

for i in $[1, N]$:

$$
\begin{aligned}
y_i &= \frac{e^{x_i-m_N}}{d'_N} \\
o_i &= o_{i-1} + y_i V[i,:]
\end{aligned} \tag{5}
$$

$$
O[k,:] = o_N \tag{6}
$$

The equation 5 equals to:

$$
\begin{aligned}
o_j &= \sum_{i=1}^{j} y_i V[i,:] \\
&= \sum_{i=1}^{j} \frac{e^{x_i - m_N}}{d'_N} V[i,:]
\end{aligned} \tag{7}
$$

As well as what we did in online softmax, we create another sequence to remove the dependency on N:

$$
o'_j = \sum_{i=1}^{j} \frac{e^{x_i - m_j}}{d'_j} V[i,:] \tag{8}
$$

when $j = N$, $o_N = o'_N$.

Then we can find the recurrence relation between o'_j and o'_{j-1}:

$$
\begin{aligned}
o'_j &= \sum_{i=1}^{j} \frac{e^{x_i - m_j}}{d'_j} V[i,:] \\
&= \sum_{i=1}^{j-1} \frac{e^{x_i - m_j}}{d'_j} V[i,:] + \frac{e^{x_j - m_j}}{d'_j} V[j,:] \\
&= \sum_{i=1}^{j-1} \frac{e^{x_i - m_{j-1} + m_{j-1} - m_j}}{d'_{j-1}} \frac{d'_{j-1}}{d'_j} V[i,:] + \frac{e^{x_j - m_j}}{d'_j} V[j,:] \\
&= \frac{d'_{j-1}}{d'_j} e^{m_{j-1} - m_j} \left(\sum_{i=1}^{j-1} \frac{e^{x_i - m_{j-1}}}{d'_{j-1}} V[i,:] \right) + \frac{e^{x_j - m_j}}{d'_j} V[j,:] \\
&= \frac{d'_{j-1}}{d'_j} e^{m_{j-1} - m_j} o'_{j-1} + \frac{e^{x_j - m_j}}{d'_j} V[j,:]
\end{aligned} \tag{9}
$$

flash attention

for i in $[1, N]$:

$$
\begin{aligned}
x_i &= Q[k,:] K^T[:,i] \\
m_i &= \max(x_i, m_{i-1}) \\
d'_i &= d'_{i-1} * e^{m_{i-1} - m_i} + e^{x_i - m_i} \\
o'_i &= \frac{d'_{i-1}}{d'_i} e^{m_{i-1} - m_i} o'_{i-1} + \frac{e^{x_i - m_i}}{d'_i} V[i,:]
\end{aligned} \tag{10}
$$

$$O[k,:] = o'_N \tag{11}$$

3 flash attention forward v1

The grid has Batch * head_num block, each block perform a single-head attention as shown as fig 1.

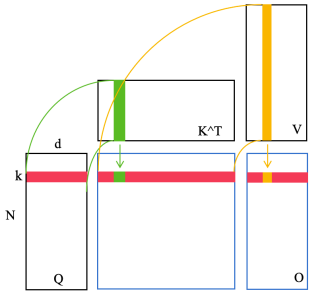

Figure 1: attention

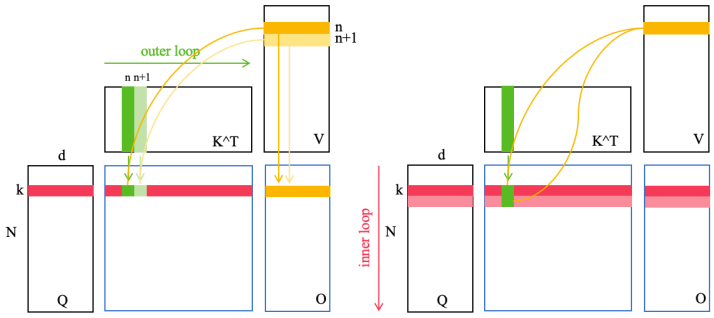

Figure 2: forward v1 traverse

At flash attention forward v1, we[1] traverse the matrix K, V in the outer layer and the matrix Q in the inner layer as shown as fig 2.

As shown as fig 3, to reduce the number of accesses to global memory, we maximize the use of shared memory and allocate it to four matrices Q_i, K_j, V_i, S as much as possible. The Q_i, S matrix acts as a temporary register, and the K_j, V_i matrix is used for intra-block thread sharing.

Then, we will update the matrix K_j, V_i in each outer loop, and update the matrix Q_i, S in each inner loop.

$$S = QK^T, \text{Attention} = \text{softmax}(\frac{S}{\sqrt{d}})V \tag{12}$$

In the block, each thread is responsible for a row in the matrix Q_i, and then traverses each column in the matrix K_j to obtain a value in S. In one step, a small section of a row in S can be obtained, we can pass online softmax method to process these small segments, but by applying this method to $\text{softmax}(S[k,:])V = O[k,:]$, we can directly calculate the matrix O through these small segments.

[1](we) means the original paper

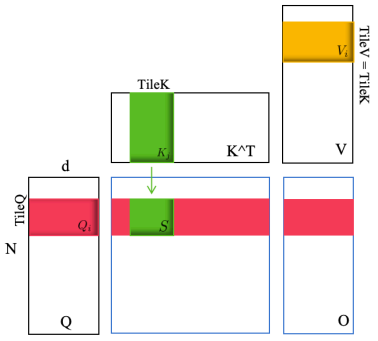

Figure 3: shared memory allocate

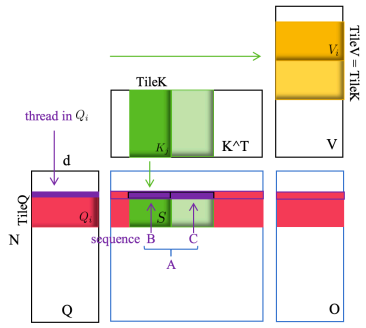

Figure 4: calculate example

As shown in the fig 4, through two outer loops, we get the statistics of the two sequences B, C, where the statistics of B were calculated last time and stored in the global memory. The statistics of C are calculated by the current loop. We maintain two arrays of size N to store the sum and max information of the sequence obtained in the last outer loop, which named l, m in original paper.

Then, for equation 8, we can generalize it so that the result of $O[k,:] = o_N^A$ can be calculated directly by two sequences B, C.

$$
\begin{aligned}
o_N^A &= \sum_{i=1}^{N} \frac{e^{a_i - A.max}}{A.sum} V[i,:] \\
&= \sum_{i=1}^{\frac{N}{2}} \frac{e^{b_i - A.max}}{A.sum} V[i,:] + \sum_{i=1}^{\frac{N}{2}} \frac{e^{c_i - A.max}}{A.sum} V[i+\frac{N}{2},:] \\
&= \frac{B.sum}{A.sum} e^{B.max - A.max} \sum_{i=1}^{\frac{N}{2}} \frac{e^{b_i - B.max}}{B.sum} V[i,:] + \frac{C.sum}{A.sum} e^{C.max - A.max} \sum_{i=1}^{\frac{N}{2}} \frac{e^{c_i - C.max}}{C.sum} V[i+\frac{N}{2},:] \\
&= \frac{B.sum}{A.sum} e^{B.max - A.max} o_{N/2}^B + \frac{C.sum}{A.sum} e^{C.max - A.max} o_{N/2}^C
\end{aligned} \tag{13}
$$

$A.sum, A.max$ can be performed by equation 3.

Generally, $o_{N/2}^B$ means the result of $O[k,:]$ after last outer loop, $o_{N/2}^C$ can be obtained with $C.max, C.sum$.

$$new.max = \max(m[k], C.max)$$
$$new.sum = l[k] \cdot e^{m[k]-new.max} + C.sum \cdot e^{C.max-new.max}$$
$$O[k,:] = \frac{l[k]}{new.sum} e^{m[k]-new.max} O[k,:] + \frac{e^{C.max-new.max}}{new.sum} \sum_{i=1}^{TileK} e^{c_i - C.max} V[i+\theta,:] \quad (14)$$

flash attention forward v1

The symbols used below are as shown in fig 3

---

for j in N/TileK do
  load K_j, V_j to shared memory
  for i in N/TileQ do
    load Q_i to shared memory
    each thread k in block perform each row in $Q_i \rightarrow Q_i[k,:]$
    for y in N/TileK do
      for x in d do
        perform $Q_i[k,x] \times K_j[x,y]$, add to sum
      end for
      perform max_tmp
      store sum to $S[k,y]$
    end for
    for y in N/TileK do
      $S[k,y] = e^{S[k,y]-\text{max\_tmp}}$
      perform sum_tmp
    end for
    load global $l[k], m[k]$ to sum_prev, max_prev
    max_new = max(max_tmp, max_prev)
    sum_new = sum_tmp $\cdot e^{\text{max\_tmp-max\_new}}$ + sum_prev $\cdot e^{\text{max\_prev-max\_new}}$
    update $l[k], m[k] \leftarrow$ sum_new, max_new
    for x in d do
      k'(offset) = $i \times TileQ + k$
      $O[k',x] = O[k',x] \times \frac{\text{sum\_prev}}{\text{sum\_new}} \times e^{\text{max\_prev}-\text{max\_new}} + \frac{e^{\text{max\_tmp}-\text{max\_new}}}{\text{sum\_new}} \times S[k,:] \cdot V_j[:,x]$
      $= \frac{1}{\text{sum\_new} \times e^{\text{max\_new}}} \times (O[k',x] \times \text{sum\_prev} \times e^{\text{max\_prev}} + e^{\text{max\_tmp}} \times S[k,:] \cdot V_j[:,x])$
    end for
  end for
end for

4 flash attention forward v2

Flash attention V2 has two main changes.

1. Swap the cyclic order for matrix Q and matrix K, V

2. Store the Logsumexp instead of l, m for backward

flash attention forward v2

The following code does not involve saving $L(logsumexp)$

---

for i in N/TileQ do
  load Qi to shared memory
  each thread k in block perform each row in $Q_i \rightarrow Q_i[k,:]$
  define max_row, sum_row for this row
  for j in N/TileK do
    load K_j, V_j to shared memory
    for y in N/TileK do
      for x in d do
        perform $Q_i[k,x] \times K_j[x,y]$, add to sum
      end for
      perform max_tmp
      store sum to $S[k,y]$
    end for
    for y in N/TileK do
      $S[k,y] = e^{S[k,y] - \text{max\_tmp}}$
      perform sum_tmp
    end for
    max_new = max(max_tmp, max_row)
    sum_new = sum_tmp $\cdot e^{\text{max\_tmp-max\_new}}$ + sum_row $\cdot e^{\text{max\_row-max\_new}}$
    for x in d do
      if j == 0 then
        //dont need to initial the O_i to zero
        $O_i[k,x] = \frac{e^{\text{max\_tmp} - \text{max\_new}}}{\text{sum\_new}} \times S[k,:] \cdot V_j[:,x]$
      else
        $O_i[k,x] = \frac{1}{\text{sum\_new} \times e^{\text{max\_new}}} \times (O_i[k,x] \times \text{sum\_row} \times e^{\text{max\_row}} + e^{\text{max\_tmp}} \times S[k,:] \cdot V_j[:,x])$
      end if
    end for
  end for
  write O_i to O
end for

flash attention forward v2 use more shared memory ($TileQ \times d$) for storing the temporary result of O.

5 Experiment

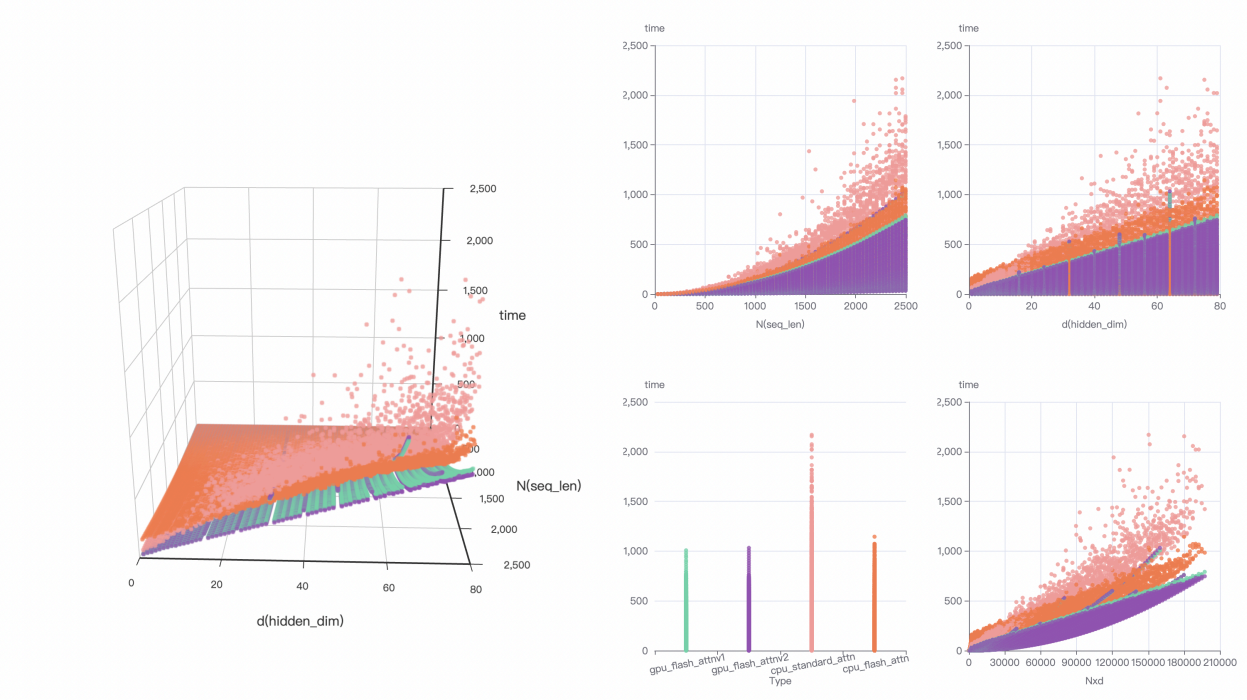

Figure 5: Profiling

As shown as fig 5, it seems that the bank conflict occurs when d is a multiple of 8, especially a multiple of 32.

The flash attention forward v2 code in this article is not very accurate because it does not achieve great acceleration, and the number of accesses to global memory $QKVO$ has nothing to do with the loop order.

More detail about profiling